RAMNARAIN_NILESH_00635805

by Nilesh Ramnarain

Harvesting Volatility Risk Premia using Deep Reinforcement Learning

September 5, 2022

A thesis presented for the degree of
Msc Mathematics & Finance 2022

Department of Mathematics
Imperial College London
Author: Nilesh Ramnarain CID: 00635805

Acknowledgements

I would like to express my gratitude to Professor Vladimir Lucic for his all of his insights and patience over the past few months. We grew quite close during this time period and his incredible support and guidance throughout this thesis was invaluable.

Additionally I would like to extend this thank you to all the lecturers and staff of the Maths Department at Imperial in being able to deliver an outstanding Masters course and teaching experience despite the covid pandemic.

Finally I wanted to thank my parents, friends and family, little Milou and my Chou for their constant love and support during the tenor of my Masters; this would not have been possible without them.

Declaration

The work contained in this dissertation is my own work unless otherwise is stated.

Contents

1 Introduction

Volatility is considered to be a somewhat tricky asset class and it's variety of connotations can often be misunderstood amongst many investors. In this research piece we aim to explore volatility strategies in order to harvest volatility risk premia (VRP); where risk premia is defined as the persistent premiums that exists in the marketplace caused by exogenous factors, which provide returns over the risk free rate.

Volatility in a simple sense, looks at the changes in an underlying asset price irrespective of the direction it takes. There are a number of derivatives that can be utilised in order to trade volatility, and we will see later on the piece how we will use some of these instruments to harvest volatility risk premia.

This is a fairly technical piece that assumes the readership audience is familiar with volatility, however due care will be taken to explicitly define terms as we go along in order to help with some of the notions. There are two key flavours of volatility, and we begin by outlining a few formal definitions below.

Definition 1. Returns - Asset price returns refer to the amount gained/lost of holding an investment over a given period of time. More formally, if we have a price series S_n where $n \in \{0, ..., N\}$ then we define the log returns R_n of the price series as follows:

$$R_n = \log S_n - \log S_{n-1} = \log \frac{S_n}{S_{n-1}}$$

Definition 2. Implied Volatility (IV) - Is the market's forecast of the likely movement in an asset price. It is a quantity that is often inferred from the market via option prices.

Definition 3. Realised Volatility (RV) - Is defined as the annualised standard deviation of returns (using a trailing window) over a given time period. This is a quantity that can be computed directly from a time series of asset price returns, taking care to scale appropriately depending on the frequency of returns.

Mathematically, suppose we have observed N asset log returns R_n where $n \in \{0, ..., N\}$ with a mean return of $\mu = \frac{1}{N}\sum_{n=1}^{N} R_n$; then the realised volatility is computed as follows:

$$\sigma_{RV} = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(R_n - \mu)^2}$$

Usually in practise, the definition of realized volatility is:

$$\sigma_{RV} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} R_n^2}$$

Where the mean return μ is often set to 0, because it's impact on price is negligible (i.e. when considering returns of an asset on a daily basis, the expected average daily return is $1/252^{nd}$ of the money-market rate.), thus the drift of the underlying asset price is assumed to be zero.

Given that volatility is quoted as an annualised number (by longstanding industry standards) appropriate scaling is taken to annualise the volatility after computation. As an example, if we consider the observation of N daily asset log returns R_n where $n \in \{0, ..., N\}$; where the asset has a 252 trading days in a year, then the annualised realised volatility is given by:

$$\sigma_{RV} = \sqrt{\frac{252}{N} \sum_{n=1}^{N} R_n^2}$$

To avoid any ambiguity, this is how we'll refer to realised volatility throughout the piece.

Definition 4. Subsequent Realised Volatility - Is defined as the annualised standard deviation of returns using a forward looking (not trailing) a given time period. This is a quantity that can be compared with the market implied volatility, retrospectively to as one form of measure to see how accurate the market was in forecasting expected volatility as of a given date.

In Chapter 1, we aim to explore and define what is meant by volatility risk premia. We aim to provide some intuitive insight into it's existence, as well as outlining a framework for harvesting volatility risk premia, by exploring the number of financial securities that can be used in order to trade volatility. Chapter 2 will outline the mathematical underpinnings of option pricing as well as numerical method techniques used for further pricing.

We then begin by exploring the foundations of reinforcement learning in Chapter 3, and outline some of the key concepts and algorithms that govern this machine learning method. We additionally discuss Neural Networks and different architectures used. We then end the section by discussing how these networks tie into Reinforcement Learning.

In Chapter 4, we spin out a mini project in order to test that we have implemented the Deep Reinforcement Learning algorithm correctly. We train an agent to hedge a European call option by learning the BS-Delta within a theoretical setting.

Finally in Chapter 5, we will introduce our back testing engine, which will utilise SPX options to harvest VRP and explore what contribution a trained deep reinforcement learning agent can make to the performance of the volatility harvesting strategy.

Volatility Risk Premia (VRP)

1.1 What is VRP?

Volatility risk premia has many different connotations, but put simply it is defined as the spread between implied volatility (IV) and realised volatility (RV) typically of an ATM option i.e:

$$VRP = \sigma_{IV} - \sigma_{RV}$$

We remark that some investors can also look at VRP along the tails (OTM options). We recall that implied volatility (IV) is annualised volatility that is observed from market prices of options and here we define realised volatility to be the subsequent annualised standard deviation of returns i.e. we subtract the realised volatility of the underlying asset returns for the subsequent period of the option tenor, from the implied volatility.

1.2 Existence of VRP

There are many explanations for the existence of VRP. One of the main drivers behind the existence of VRP comes from the fact that option sellers wish to be compensated for large but infrequent losses that may occur. E.g. sellers of put options on the SPX demand a higher premium in the event of a gap move to the downside if the US equity market suffers a crash. Another reason that VRP exists arises due to supply/demand imbalances that occur in the marketplace. I.e. there is a a demand for OTM options as they are in fact cheaper than buying ATM options and this also leads to the so-called "volatility smile" or skew as we will refer to the shape of the smile.

We illustrate the existence of VRP with an example on the S&P 500. VRP in this example is computed as a ratio of rolling 1M ATM IV/1M subsequent RV, which we see in Figure 1 below.

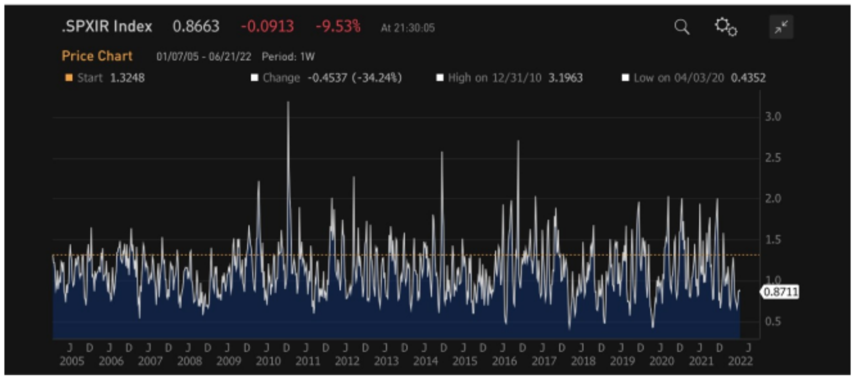


Figure 1: 1M Implied to Realised ratio of Volatility in the S&P 500 options market, Source: Bloomberg Market Data

When this ratio is above 1, we have existence of VRP and the mean of this spread over the last 20 years is 1.4 which gives us a leading indication that there is VRP in the options market which can be harvested here. We remark that this spread is not a directly tradeable instrument and there are a number of ways one can trade this spread accurately which will be discussed in the next section.

1.3 How do we harvest VRP?

There are many different instruments one can use to harvest VRP. Some of these methods range from selling listed option strategies (i.e. selling straddles, strangles, or even outright put/call strips) to OTC-swap based strategies involving Volatility and Variance Swaps.

Exposure to VRP and the efficiency of extracting it can vary significantly between different methods, factoring in not only market accessibility but also operation efficiency of maintaining the positions as well as considering investor risk profiles.

There are a number of pros and cons associated with each of the implementations and can play a big role in the choice that an investor decides to make. We aim to briefly summarise the pros and cons associated to each method of harvesting VRP in the table below.

Instruments	**Risk premia exposure**	**Conditions for +ive P&L**	**Pros**	**Cons**
Variance swaps	Implied-realised variance (vol squared)	Implied VarStrike > RV	Clean exposure to VRP; higher strike than Volatility swaps	OTC (non-transparent); convexity exposure to vol spikes
Volatility swaps	Implied-realised volatility	Implied VolSwap strike > RV	Clean exposure to VRP	OTC (non-transparent); very heavily model dependent; lower strike than VarSwaps
Delta-hedged straddles/strangles	Path dependent implied-realised vol (scaled by gamma weighted RV)	Low RV around the strike (gamma highest in this region)	Tailored (delta-hedging program) & Targeted strike exposure to VRP (custom strikes can be traded); listed instruments are more liquid across asset classes	Imperfect exposure to VRP as options are path dependant; additional costs of delta-hedging execution
Non delta-hedged straddles/strangles	IV-RV	Underlying cannot deviate outside the strike range	Targeted strike exposure (custom strikes can be traded); listed instruments are more liquid across asset classes	Indirect exposure to VRP (P&L purely dependant on delta, i.e. where underlying is trading at expiry)
Listed VarSwap Replication	Implied-realised variance (vol squared)	IV > RV, provided underling remains within chosen strike range	Clean exposure to VRP using listed, liquid instruments	Operationally intensive; lower VRP than OTC VarSwaps

Table 1: Methods of Harvesting VRP

In general, Volatility & Variance Swaps provide the cleanest and most direct exposure to harvest VRP; whereas delta-hedged option strategies have the drawback of path-dependency and are a lot more operationally intensive. Option strategies however take great benefit from being listed instruments, which in general are more liquid than their OTC counterparts and also allow for a greater customisation of VRP exposure.

In the later stages of the report, Variance Swaps (VarSwaps) will be discussed in further detail, providing additional context to their mathematical preliminaries as well as how one can go about static replication of a VarSwap using listed options; a key strategy we will be using in order to harvest VRP.

2 Mathematical Preliminaries

2.1 Black-Scholes Pricing Model

In this chapter, we aim to provide the fundamental basics of the famous Black-Scholes pricing framework. As usual we begin in a risk neutral setting using a filtered-probability space $(\Omega, \mathcal{F}_t, \mathbb{Q})$ and assume that an underlying stock price S_t follows a Geometric Brownian Motion i.e.

$$dS_t = rS_t dt + \sigma S_t dW_t^{\mathbb{Q}}$$

Where $W_t^{\mathbb{Q}}$ is a standard Brownian Motion under risk-neutral measure $\mathbb{Q}$. We recall that the price a European Call option is given by:

$$C(S,t) = \mathbb{E}_t^{\mathbb{Q}}[(S_T - K)^+]$$

Where S_T denotes the asset price at expiry. From lectures, we have seen that the above can be solved to yield the following call price at time t:

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\log\frac{S_t}{K} + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

$$C(S,t) = S_t N(d_1) - Ke^{-r(T-t)}N(d_2)$$

Where N(.) is the standard cumulative normal distribution function:

$$N(x) = \mathbb{P}(X \leq x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{-x^2}{2}}dx$$

Similarly recalling the relationship of put-call parity for European options, one can immediately deduce the price of the put as follows:

$$C(S,t) - P(S,t) = S_t - Ke^{-r(T-t)} \implies P(S,t) = S_t - Ke^{-r(T-t)} + C(S,t)$$

$$P(S,t) = Ke^{-r(T-t)}N(-d_2) - S_t N(-d_1)$$

Additionally, we remark that the above equations for the call/put price can also be derived from the famous Black-Scholes PDE i.e.

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + rS\frac{\partial C}{\partial S} - rC = 0$$

BS Delta

An option's delta represents the option premium's sensitivity to the underlying asset. It can be found by taking the partial derivative of the option price w.r.t. the underlying asset S_t. From this we can deduce that the deltas for calls and puts respectively are as follows:

$$\Delta_{call} = \frac{\partial C(S,t)}{\partial S} = N(d_1)$$

$$\Delta_{put} = \frac{\partial P(S,t)}{\partial S} = -N(-d_1) = N(d_1) - 1 = \Delta_{call} - 1$$

BS Gamma

Option gamma is a second order greek and represents an option delta's sensitivity to the underlying asset. It can be derived by taking the partial derivative of an option's delta w.r.t. the underlying asset. It can be shown that an option's gamma satisfies:

$$\Gamma = \frac{\partial \Delta}{\partial S} = \frac{N'(d_1)}{S\sigma\sqrt{T-t}}$$

BS Vega

Option vega here measures the sensitivity of an option's price w.r.t. a 1% change in implied volatility. It can be derived by taking the partial derivative of an option price w.r.t. σ and it can be shown that option vega satisfies:

$$\nu = \frac{\partial C(S,t)}{\partial \sigma} = \frac{\partial P(S,t)}{\partial \sigma} = SN'(d_1)\sqrt{T-t}$$

Newton-Rhapson Root Solving for IV

Given a series of option prices, it is often useful to deduce what the IV is of the option. In practise this is done via a Newton-Rhapson approach and we aim to show the steps involved in extracting the IV numerically. This is often done with an initial guess to what the IV is which we will denote as $\tilde{\sigma}$. WLOG we consider "backing out" the IV for a call option as follows. Here we parameterise the option price as a function of volatility σ and we recall that the price of a call option using our initial guess of IV is given by:

$$C(\tilde{\sigma}) = SN(d_1) - Ke^{-r(T-t)}N(d_2)$$

Where in the usual sense d_1 and d_2 are defined above only this time σ is replaced with $\tilde{\sigma}$.

We also recall that call option vega is given by the following expression:

$$\nu = C'(\tilde{\sigma}) = SN'(d_1)\sqrt{T-t}$$

Now via a first order Taylor series expansion between the actual IV σ and the initial guess $\tilde{\sigma}$ we deduce the following expression:

$$\begin{aligned} C(\sigma) &= C(\tilde{\sigma}) + C'(\tilde{\sigma})\delta\tilde{\sigma} \\ &= C(\tilde{\sigma}) + C'(\tilde{\sigma})(\sigma - \tilde{\sigma}) \\ &= C(\tilde{\sigma}) + C'(\tilde{\sigma})\sigma - C'(\tilde{\sigma})\tilde{\sigma} \end{aligned} \tag{2.1.1}$$

Re-arranging this equation to solve for the true IV i.e. σ we deduce that:

$$\sigma = \frac{C(\sigma) - C(\tilde{\sigma}) + C'(\sigma)\tilde{\sigma}}{C'(\tilde{\sigma})}$$

By iterating through this expression until σ is closely approximated by $\tilde{\sigma}$ given some small epsilon error threshold we can deduce the IV of the option.

Inverting BS Delta-strike to Fixed-strike

It is often industry standards to quote delta-strikes for options as opposed to fixed numerical strikes. This is because as the underlying forwards move over time, one needs to keep track of the fixed strike and it's distance away from the underlying forward which makes it difficult when comparing option strikes of two different assets.

For this reason, the delta-strikes are commonly used in practise as the "moneyness" or (relative position of forward to strike) is embedded in the delta; making it an easy way to compare option strikes even across asset classes. Given a call delta-strike Δ_c, a simple inversion of the BS Delta formula means we can back out the fixed strike algebraically i.e.

$$\Delta_{call} = N(d_1) = N\left(\frac{1}{\sigma\sqrt{T-t}}\left[\log\frac{S_t}{K} + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]\right)$$

Re-arranging the above formula for the call fixed strike "K" we obtain that:

$$K_c = \frac{S}{\exp\left[\sigma\sqrt{T-t}N^{-1}(\Delta_c) - (r+\frac{1}{2}\sigma^2)(T-t)\right]}$$

And via put-call parity for option deltas, we can deduce that the put fixed strike "K" as:

$$K_p = \frac{S}{\exp\left[\sigma\sqrt{T-t}N^{-1}(\Delta_p+1) - (r+\frac{1}{2}\sigma^2)(T-t)\right]}$$

We remark that the fixed strike obtained from this re-arranging may not be a market tradeable fixed strike. In practise, the fixed strike obtained is then rounded to the nearest strike using the option strike contract specifications and the minimum strike increment.

In practise, under a flat vol skew assumption (i.e. implied volatility is constant across all strikes) the σ_{ATM} can be used to proxy what these strikes will be. Generally speaking this technique works well in FX options which tend to exhibit the flattest skew smiles; however steep put skew is often pronounced in equities, meaning that this approach of using the ATM IV may not be suitable when backing out the strike from delta.

Decomposing option P&L

We recall that a European call option is of the form $C(S,\sigma,t)$. By applying Ito's lemma to this expression we deduce the following:

$$\begin{aligned} dC &= \frac{\partial C}{\partial S}dS + \frac{1}{2}\frac{\partial^2 C}{\partial S^2}dS^2 + \frac{\partial C}{\partial t}dt + \frac{\partial C}{\partial \sigma}d\sigma + \ldots \\ &= \Delta_c ds + \frac{1}{2}\Gamma dS^2 + \theta_c dt + \nu d\sigma + \ldots \\ &= \text{Delta PnL} + \text{Gamma PnL} + \text{Theta PnL} + \text{Vega PnL} + \ldots \end{aligned} \tag{2.1.2}$$

Now recall from option pricing theory that if we have a portfolio Π made up of a long delta-hedged call option i.e. $\Pi = C(S,t) - \Delta_c S$ then looking at the instantaneous change in this portfolio we deduce that $d\Pi = dC - \Delta_c dS$. Thus we can deduce that the change in option price (i.e. PnL change) can be decomposed as follows:

$$\text{Delta-hedged Call PnL} = \text{Gamma PnL} + \text{Theta PnL} + \text{Vega PnL} + \ldots \tag{2.1.3}$$

We leave this expression for PnL above, as it will become useful when discussing the static replication of Variance swaps in the next section.

Harvesting VRP via Delta-Hedged Options

In this section we briefly aim to discuss how we can harvest VRP through selling delta-hedged options. By assuming that the stock price S_t follows the GBM:

$$dS_t = rS_t dt + \sigma S_t dW_t^{\mathbb{Q}}$$

And by denoting call option price $C(S_t, t)$, one can apply Ito's formula to deduce that:

$$C_t - C_0 = \int_0^t \frac{\partial C_u}{\partial S_u} dS_u + \int_0^t \left(\frac{\partial C_u}{\partial u} + \frac{1}{2}\sigma^2 S_u^2 \frac{\partial^2 C_u}{\partial S_u^2} \right) du$$

We can replace the expressions in the integral with terms from the Black-Scholes PDE to deduce:

$$C_t = C_0 + \int_0^t \Delta_u dS_u + \int_0^t r\left(C_u - \Delta_u S_u\right) du$$

We now consider a portfolio Π_t consisting of a hedged call option struck at K i.e. $C(K, t, \tau)$ with corresponding option delta $\Delta_t = \Delta(K, t, \tau)$ units of the underlying stock S_t. The PnL change of the delta-hedged portfolio Π_t between $(t, t+\tau)$ can be defined as:

$$\Pi_{t,t+\tau} = C_{t+\tau} - C_t + \int_t^{t+\tau} \Delta_u dS_u + \int_t^{t+\tau} r\left(C_u - \Delta_u S_u\right) du$$

i.e. this governs the portfolio PnL of the delta-hedged option; where the net cash earns the risk-free rate. The expected value of this expression as highlighted in [4, Bakshi & Kapadia, 2003] corresponds to the expected excess rate of return for the delta-hedged portfolio Π_t.

We now consider being short delta-hedged options with implied volatility $\tilde{\sigma}$. It can be shown that by substituting the Black-Scholes PDE with the new implied volatility, into the PnL change of the delta-hedged portfolio and applying Ito's lemma that the VRP harvested can be deduced as:

$$VRP_t = \frac{1}{2}\int_0^t \Gamma(u, S_{u-}) S_u^2 \left(\tilde{\sigma}^2 - \sigma^2\right) du$$

We remark that the quantity $\frac{1}{2}\Gamma S_t^2$ is the total dollar gamma of the portfolio and a short position in options allows for a net positive portfolio gamma a.s.

2.2 Variance Swap Introduction

In this brief section closely follow the work done in the paper by [2, J.P. Morgan]; we outline a few preliminaries and background to variance swaps as well as their synthetic replication using options. We introduce this section because one of the ways we will be harvesting VRP will be done through VarSwap replication through option strips.

Definition 5. Variance swap - A variance swap is a derivative in similar fashion to a forward contract, which allows an investor to trade realized variance against a pre-determined strike price (where the current IV^2 forms the strike price of the variance swap).

The final payoff of the variance swap is given as follows:

$$VarSwap_{payoff} = \text{Variance Notional} \times (FinalRV^2 - Strike^2)$$

Where the variance notional is given by:

$$\text{Variance Notional} = \frac{\text{Vega Notional (in USD)}}{2 \times Strike}$$

We remark that the payoff of a VarSwap is convex in volatility as illustrated in the plot below. Mathematically we deduce this from Jensen's inequality where we recall that:

$$\mathbb{E}\left[\sqrt{\text{Variance}}\right] \leq \sqrt{\mathbb{E}\left[\text{Variance}\right]}$$

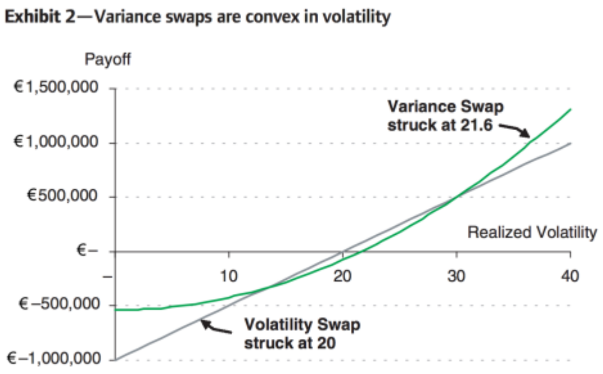


Figure 2: Variance swap payoff example. Image taken from [1]

More on Delta-hedged Option P&L

We have previously showed that delta-hedged option PnL can be broken down and expressed as follows:

$$\begin{aligned} dV &= \frac{1}{2}\Gamma dS^2 + \theta dt + \nu d\sigma + ... \\ &= \text{Gamma PnL} + \text{Theta PnL} + \text{Vega PnL} + ... \end{aligned} \tag{2.2.1}$$

In the absence of stochastic volatility (i.e. assuming that implied volatility is constant) and in a world where riskless interest rates are zero; we have that the daily option P&L can be expressed as follows:

$$\begin{aligned} dV &= \frac{1}{2}\Gamma dS^2 + \theta dt \\ &= \text{Gamma PnL} + \text{Theta PnL} \end{aligned} \tag{2.2.2}$$

Assuming that there are "no free-lunches" we should have that the PnL = 0 i.e. we can equate:

$$-\frac{1}{2}\Gamma dS^2 = \theta dt \tag{2.2.3}$$

Taking the expectation of both sides i.e.

$$-\mathbb{E}\left[\frac{1}{2}\Gamma dS^2\right] = \mathbb{E}\left[\theta dt\right] \tag{2.2.4}$$

Yields the well known expression that links together both Gamma and Theta i.e.

$$-\frac{1}{2}\Gamma\sigma^2 S^2 dt = \theta dt \tag{2.2.5}$$

i.e.

$$\theta \approx -\frac{1}{2}\Gamma\sigma^2 S^2 \tag{2.2.6}$$

Where S is the current underlying spot price level and σ is the current IV of the option. By writing $(\Delta S)^2 = \left(\frac{\Delta S}{S}\right)^2 S^2$, and substituting the expression for theta back into equation (5), we can express the daily PnL in terms of the squared returns and squared IV i.e.

$$\text{Daily PnL} = \frac{1}{2}\Gamma S^2 \times \left[\left(\frac{\Delta S}{S}\right)^2 - \sigma^2 \Delta t\right] \tag{2.2.7}$$

Now we focus our attention on the first term i.e. $(\frac{\Delta S}{S})$ can be thought of as % returns i.e. daily percentage changes in the stock price; thus squaring this quantity can be thought of as realised variance over one day. The second term $\sigma^2\Delta t$ can be thought of as the daily implied variance (since we are squaring the volatility).

Summing together all of the daily PnL's per day until option maturity means we can deduce an expression for the final PnL of the option i.e.

$$\text{Final PnL} = \frac{1}{2}\sum_{t=0}^{n} \gamma_t \left[r_t^2 - \sigma^2 \Delta t\right] \tag{2.2.8}$$

The expression above tells us that the final PnL of a delta-hedged option is closely related to the spread between realised-implied variance. It very closely resembles the payoff of a variance swap, with the exception that the weights in the expression above depend on the option's gamma through time; whereas in the variance swap the weights are all constant.

Static Replication of VarSwaps Using Options

Previously we have seen that a delta-hedged option strategy essentially replicates the payoff of a variance swap which is weighted by the options dollar gamma. In this section we aim to remove this weighting to have a 'non-gamma-weighted' variance swap replication method using options.

We begin by exploring what the dollar gamma looks like as a function of the underlying forward level (i.e. by strikes); we demonstrate this with a plot below using an arbitrary strike range:

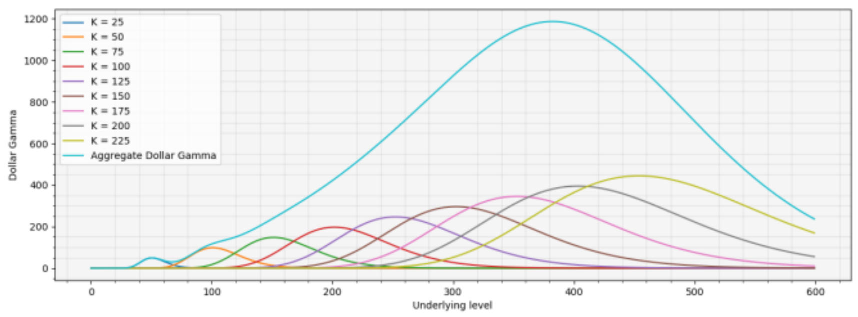


Figure 3: Option Dollar Gamma w.r.t. Underlying Level

From the above plot, we can see that the dollar gamma of the options is proportional to the underlying level. Hence, our first intuition into "flattening" the aggregated gamma across the underlying strikes is to proportionally weight higher strikes with a larger weighting and vice versa for smaller strikes. i.e. increase the weights for lower strike options and increase the weighting for higher strike options.

We now aim to prove a proposition that shows flat gamma can be achieved with a weighting that is proportional to $\frac{1}{K^2}$

Proposition 1. (Replicating State-Contingent Payoffs with Options) Given European options C, P (priced under some pricing formula) and for simplicity we assume that the state of an economy is distinguished by stock index level S only and let $p(S_T, T; S_t, t)$ denote the state-price given stock level S_T and time T.

Suppose we have function g which is twice continuously differentiable, then the value of a state-contingent claim[1] with payoff $g(S_T)$ satisfies:

$$\mathbb{E}_t(g(S_T)|S_t) = g(F) + \int_0^F g''(K)P(S_t, K, T)dK + \int_F^\infty g''(K)C(S_t.K, T)dK \qquad (2.2.9)$$

[1]Further details regarding contingent claims can be found in the original paper [Breeden & Litzenberger (1978)] Prices of State Contingent Claims Implicit in Option Prices

Proof. We denote T as the expiry of the contract with $0 \leq t < T$; then the state-price satisfies:

$$p(S_T, T; S_t, t) = \frac{\partial^2 P(S_t, K, T)}{\partial K^2} \mid_{S_T=k} = \frac{\partial^2 C(S_t, K, T)}{\partial K^2} \mid_{S_T=k} \tag{2.2.10}$$

Where $C(S_t, K, T)$ and $P(S_t, K, T)$ represent option prices; and the state-price can be derived by taking the second partial derivative of each of the option prices w.r.t. the strike price K. Hence the value of a state-contingent claim with payoff $g(S_T)$ satisfies:

$$\begin{aligned} \mathbb{E}_t(g(S_T)|S_t) &= \int_0^\infty g(K) p(K, T; S_t, t) dK \\ &= \int_0^F g(K) \frac{\partial^2 P(S_t, K, T)}{\partial K^2} dK + \int_F^\infty g(K) \frac{\partial^2 C(S_t, K, T)}{\partial K^2} dK \end{aligned} \tag{2.2.11}$$

Where F is the T-forward price of the index at time t. Integrating the above by parts yields:

$$\begin{aligned} \mathbb{E}_t(g(S_T)|S_t) &= g(K) \frac{\partial P(S_t, K, T)}{\partial K} \mid_0^F - \int_0^F g'(K) \frac{\partial P(S_t, K, T)}{\partial K} dK \\ &+ g(K) \frac{\partial C(S_t, K, T)}{\partial K} \mid_F^\infty - \int_F^\infty g'(K) \frac{\partial C(S_t, K, T)}{\partial K} dK \\ &= g(F) - \int_0^F g'(K) \frac{\partial P(S_t, K, T)}{\partial K} dK - \int_F^\infty g'(K) \frac{\partial C(S_t, K, T)}{\partial K} dK \end{aligned} \tag{2.2.12}$$

Finally we apply integration by parts a second time to yield:

$$\begin{aligned} \mathbb{E}_t(g(S_T)|S_t) &= g(F) - g'(K) P(S_t, K, T) \mid_0^F + \int_0^F g''(K) P(S_t, K, T) dK \\ &- g'(K) C(S_t, K, T) \mid_F^\infty + \int_F^\infty g''(K) C(S_t, K, T) dK \\ &= g(F) + \int_0^F g''(K) P(S_t, K, T) dK + \int_F^\infty g''(K) C(S_t, K, T) dK \end{aligned} \tag{2.2.13}$$

The above equation tells us that any twice continuously differentiable function g due at time T may be replicated by an infinite strip of call and put options with maturity at time T. Ofcourse in practise this is not feasible as the exchange has a limited number of strikes that are available for trading as well as liquidity constraints with deep OTM options. □

Our goal is to flatten out the total dollar gamma profile when across the different strikes. If we were to assume a constant dollar gamma derivative exists; how we may go about finding such a derivative? From the definition of dollar gamma for this derivative g we have:

$$\Gamma_{\$}(F) = \frac{\partial^2 g}{\partial F^2} F^2 \tag{2.2.14}$$

Where g, F are the prices of the derivative and underlying respectively. A constant dollar gamma would imply that for some constant "c" we have:

$$\frac{\partial^2 g}{\partial F^2} = \frac{c}{F^2} \tag{2.2.15}$$

Solving the above PDE yields the solution:

$$g(F) = -c\log(F) + dF + e \text{ where } c, d, e \in \mathbb{R} \tag{2.2.16}$$

Where we can interpret the above derivative with constant dollar gamma as being a log contract, along with the underlying forward and some units of residual cash. We now introduce the notion of log contracts more formally below.

We recall that long positions in variance at maturity pay the difference between the fixed implied variance strike and the realised variance over the lifetime of the swap. We now introduce log-contracts and these derivatives are crucial when it comes to the pricing of variance swaps. We consider the log-contract with payoff $g(S_T)$ as follows:

$$g(S_T) = \log\left(\frac{S_T}{K}\right) \tag{2.2.17}$$

With partial derivatives satisfying:

$$g'(S_T) = \frac{1}{S_T} \tag{2.2.18}$$

$$g''(S_T) = -\frac{1}{S_T^2} \tag{2.2.19}$$

Using the method of option spanning in Proposition 1 above, we have that the price of the log contract is as follows:

$$\begin{aligned}\mathbb{E}\left(\log\left(\frac{S_T}{F}\right)|S_t\right) &= \log\frac{F}{F} + \int_0^F -\frac{1}{K^2}P(S_t,K,T)dK + \int_F^\infty -\frac{1}{K^2}C(S_t,K,T)dK \\ &= -\int_0^F \frac{P(S_t,K,T)}{K^2}dK - \int_F^\infty \frac{C(S_t,K,T)}{K^2}dK\end{aligned} \tag{2.2.20}$$

Hence we deduce that the log contract can be replicated using an infinite number of option strips (both calls and puts) where each individual option is weighted by $\frac{1}{K^2}$; this provides us with some intuition of a sizing factor we can apply to "flattening" out the aggregated dollar gamma across the strikes we had above to remove the "dollar-gamma" weight per strike.

We applied this weighting to see it's effects on the total aggregated dollar gamma across strikes using the arbitrary strike range before and the results of the weighting are illustrated below:

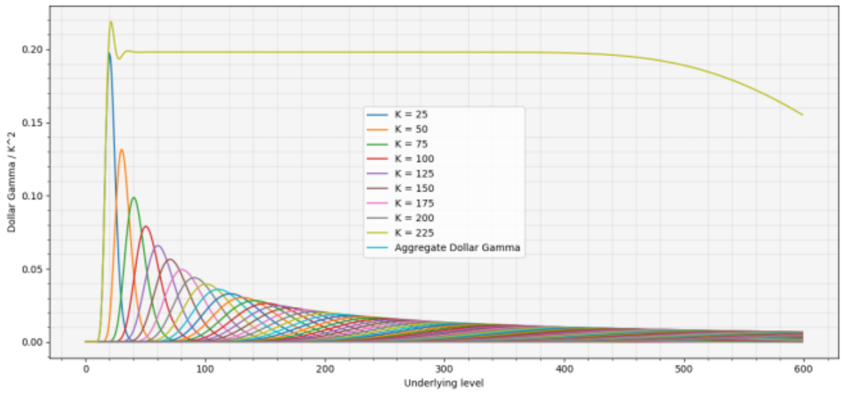


Figure 4: Option Dollar Gamma w.r.t. Underlying Level weighted by $\frac{1}{K^2}$

From the above plot, we have gained a linear region of flat gamma across the strikes using this weighting. As mentioned earlier, in practise it is impossible to trade an infinite number of strikes; however this result is fundamental when it comes the valuation of constructing a synthetic variance swap with options.

For completeness we add a reference to the $\$Gamma$ interpretation from a truncated BS-Log Contract above [6, Volatility Notes]. We start off by taking a log contract $V \in \{Put, Call\}$ and differentiating this twice yields the following expression (under flat volatility skew assumption):

$$F^2 \frac{\partial^2 V}{\partial F^2} = K^2 \frac{\partial^2 V}{\partial K^2}, \quad V \in \{\text{Put}, \text{Call}\}. \tag{2.2.21}$$

We have, with F denoting the market forward, the following expression for the truncated Log Contract on $[L, U]$, $L < F < U$,

$$L(F; L, U) = \int_L^F \frac{\text{Put}(K,T)}{K^2}\, dK + \int_F^U \frac{\text{Call}(K,T)}{K^2}\, dK,$$

$$\text{BSTruncatedLogContract\$Gamma} := F^2 \frac{\partial^2 L}{\partial F^2}(F; L, U) = \int_L^U p(K)\, dK = \mathbb{P}\big(S_T \in [L, U]\big),$$

where $p(\cdot)$ is the probability density of the underlying. We conclude that for $L = 0$, $U = \infty$ we recover flat \$Gamma equal to one, as expected, but also that truncation of the region of integration has the effect of reducing exposure by $\mathbb{P}\big(S_T < L\big) + \mathbb{P}\big(S_T > U\big)$.

3 Elementary Foundations of Reinforcement Learning

3.1 Introduction to Reinforcement Learning

We begin this chapter by providing an elementary grounding in the concepts of Reinforcement Learning. The topics discussed in this introduction closely follow the Reinforcement Learning textbook from Sutton & Barto [9].

Reinforcement Learning simply put is the training of an agent's decision making process within a specified environment where the goal of the agent is to take a series of actions in order to maximise his expected reward. The rewards setup can be done in such a way that the agent either chooses to maximise it's success, either in the subsequent long or short term.

The challenging part of RL comes in the design of the framework, i.e. defining the environment, states, actions and rewards that an agent can perform. The flexibility and range of choices contribute to the powerful applications of RL. We begin by providing a simple overview of the baseline components used in the RL paradigm:

1. Agent: An RL agent is the entity which we are training to make correct decisions.
2. Environment: The RL environment is the surroundings in which the RL agent interacts with. The agent can only control it's own actions and cannot manipulate the environment.
3. State: States define the current situation that the agent finds itself in.
4. Action: The choice that the agent makes at the current time step. The set of actions that an agent can take a known as a priori and the feasible action that an agent takes may subsequently affect the state that the environment transitions to.
5. Reward: The reward signal defines the main goal of the RL problem. At each time step following an agent's action, the environment sends the agent a single number called the reward. The goal of the agent is to maximise it's set of rewards over some time horizon and must be able to distinguish between good and bad events.
6. Policy: A function that maps the agent's states to actions.

In order to build an optimal policy (i.e. sequence of decision making) the agent must make decisions trading off between the reward signals it obtains as well as exploring new states. This trade-off is known as Exploration vs. Exploitation. In order to strike a right balance, the best overall strategy may involve short term sacrifices hence an agent should collect enough information to make the best decision in the future.

3.2 Markov Decision Processes

We introduce this section on finite Markov Decision Processes (MDP's) as they form the mathematical basis for RL problems. MDP's model decision making in discrete, stochastic environments.

At each time step, the agent observes the state S_t of the environment, chooses an appropriate action $A_t \in \mathcal{A}(s)$, receives reward $R_{t+1} \in \mathcal{R}$ and subsequently transitions to the next state $S_{t+1} \in \mathcal{S}$. This sequential process of state, action, reward, next state, next action is often referred to as SARSA (i.e. $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, ..$). The following diagram taken from [9] depicts the sequential decision making:

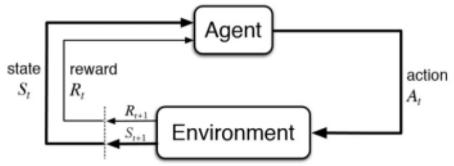


Figure 5: Agent-environment interaction in MDP setting

In the setting of a finite MDP, the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ have a finite number of elements and have well defined discrete probability distributions. We can define the dynamics of this distribution depending solely on the preceding state and action with $\mathcal{S}, \mathcal{R}$ as random variables as follows:

$$p(s', r|s, a) = \mathbb{P}(S_t = s', R_t = r, |S_{t-1} = s, A_{t-1} = a) \tag{3.2.1}$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$. We remark that as p defines a probability distribution, we must have that the following holds:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \tag{3.2.2}$$

for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$.

3.3 Optimal Policies and Rewards

We recall that the objective of the RL agent is to learn and maximise the cumulative reward it receives. Given a sequence of rewards $R_{t+1}, R_{t+2}, R_{t+3}$ we define the discounted expected rewards after time t as follows:

$$G_t = R_{t+1} + \gamma G_{t+1} = \sum_{i=0}^{T} \gamma^i R_{t+i+1} \tag{3.3.1}$$

Where discount factor γ is a parameter satisfying $0 \leq \gamma \leq 1$. The discount rate is a factor in determining the present value of future rewards and at the limits, if γ is close to 0, then the agent seeks to maximise it's immediate returns and if γ is closer to 1, the agent seeks to maximise it's longer term rewards.

We recall previously that the mapping between states and actions is called a policy. To define this more formally, a policy is a mapping from states to probabilities of selecting each possible action. An agent that follows policy π given that he is in state s, chooses an action a with probability $\pi(a|s)$. For deterministic policies, we denote $\pi(s)$ as the action chosen, given the starting state s.

We now proceed to defining the state-value function and action-value function which are fundamental in Reinforcement Learning.

State-Value Function
The state-value function of a state s following policy π is given by:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1}|S_t = s\right] \tag{3.3.2}$$

We remark that the value of the terminal state (if any) is always 0.

Action-Value Function
The action-value function of taking action a starting in state s following policy π is given by:

$$q_\pi(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1}|S_t = s, A_t = a\right] \tag{3.3.3}$$

From the recursive nature of these functions, one can easily deduce the Bellman Equations as follows:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_t + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s,a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s,a)\left[r + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a')\right] \end{aligned} \tag{3.3.4}$$

and similarly for the the state-value function we have:

$$v_\pi(s) = \mathbb{E}_\pi[R_t + \gamma G_{t+1}|S_t = s] = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s,a)[r + \gamma v_\pi(s')] \tag{3.3.5}$$

A policy π is defined to be better than another policy π' if $\pi \geq \pi'$ iff $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. The policy that is better than or equal to all other policies is denoted by the optimal policy π_*. Under the optimal policy we have the optimal state-value function and optimal action-value function as follows:

$$v_*(s) = \max_\pi v_\pi(s) \tag{3.3.6}$$

for all $s \in \mathcal{S}$ and

$$q_*(s) = \max_\pi q_\pi(s,a) \tag{3.3.7}$$

for all $s \in \mathcal{S}, a \in \mathcal{A}$. Using the optimal policy value functions above, we can deduce the set of Bellman-Optimality equations as follows:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) = \max_a \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s,a)[r + \gamma v_{\pi_*}(s')] \tag{3.3.8}$$

$$q_*(s,a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s,a)[r + \gamma \max_{a'} q_*(s', a')] \tag{3.3.9}$$

3.4 Policy Improvement and Iteration

By using the Bellman equation for v_π as an update rule for a sequence of value functions $v_0, v_1, v_2, ..$ we have the following iterative representation for v_π as follows:

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma v_k(s')] \tag{3.4.1}$$

Where the initial state value v_0 is chosen arbitrarily. For simple episodic problems the sequence $\{v_k\}$ can be shown to converge to v_π as $k \to \infty$. One way of optimising a certain policy π given state s, is by changing action $a \neq \pi(s)$ and observing whether $q_\pi(s, a) \geq v_\pi(s)$. If this relation holds, then selecting action a yields a better value function that following the original policy π. This process is called the policy improvement theorem.

Theorem 1. (Policy Improvement) Let π and π' be any pair of policies such that:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

holds for all $s \in \mathcal{S}$. Then policy $\pi' \geq \pi$.

We now introduce the notion of a greedy policy π'. It is a natural consideration to consider changes at all states to all possible actions that appear best according to $q_\pi(s, a)$. We do this as follows:

$$\begin{aligned}
\pi'(s) &= \arg\max_a q_\pi(s, a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned} \tag{3.4.2}$$

The greedy policy π' is at least as good as the original policy π. This process of changing the original policy to be greedy w.r.t. the action value function is known as policy improvement; which adheres to the policy improvement theorem. Policy improvement and policy evaluation can be combined together to yield policy iteration. The way in which this works is policy evaluation improves the approximation in the value function $v_{\pi n}$ and policy improvement improves policy π. The following sequence diagram depicts this:

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 ... \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_* \tag{3.4.3}$$

Where I represents policy improvement and E depicts policy evaluation. Further details can be found in [9, Chapter 4.3, Pg 78-80].

3.5 Temporal Difference Learning

We remark that the update rule given above is computationally inefficient as it involves sweeping through each of the states as well as the actions one can take in that state. For a small number of states and action sets this may be feasible; however for more complex problems the computational time grows exponentially.

Instead we now consider Temporal Differencing (TD) methods; which update current estimates of the value function based on previously learned values and unlike Monte Carlo methods (more on this in [9, Chapter 5]) which require the end of the episode to be reached in order to make an update to $V(S_t)$; TD methods are only required to wait until the next time step $t+1$. A one step TD(0) update can be written as:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3.5.1}$$

We note that this transition makes the update to state S_{t+1} whilst receiving reward R_{t+1}. This is a special case of TD(λ) which can be found in further detail in [9, Chapter 7, 12]. We also note that the there exists a corresponding algorithm for action-value function which is as follows:

$$QS_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{3.5.2}$$

This update is done after every transition from a non-terminal state S_t. If S_{t+1} is in fact a terminal state then we have that $Q(S_{t+1}, A_{t+1}) = 0$. This update rule uses the tuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ that makes up a transition from one state-action pair to the next one; and gives rise to the name SARSA for the algorithm. It is an on-policy algorithm, which means that it tries to evaluate/improve the current policy that is being considered.

3.5.1 Q-Learning: Off Policy TD Control

We state the update rule for an off-policy TD(0) control algorithm as follows:

$$QS_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3.5.3}$$

We recall that an off-policy algorithm aims to evaluate/improve a policy different from that used to generate the data. The learned action-value Q aims to directly approximate q_* independent of the policy that is followed. It can be shown under certain assumptions that Q converges to q_* with probability 1. We outline the pseudo code for the algorithm below:

Algorithm 1: Q-Learning (off policy TD control) for estimating $\pi \approx \pi_*$

```
1  Algorithm parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$;
2  Initialise Q(s,a) for all s $\in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, .) = 0$
3  foreach episode do
4      Initialize S;
5      while step of episode do
6          Choose A from S using policy derived from Q (i.e. $\epsilon-$greedy;
7          Take action A and observe R, S';
8          Apply update rule:
            $QS_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$;
9          Update $S \leftarrow S'$;
10     Until S is terminal;
```

3.6 Deep Q-Learning

So far we have only considered tabular methods for reinforcement learning as states and state-action pairs were mapped to singular values. This indirectly assumed that the cardinality of states we were considering $|\mathcal{S}|$ was finite. In a trading setup this may not be entirely accurate, and there may be infinitely many states attainable.

In this section we aim to introduce a continuous setting in which we can handle infinitely many states, which is attained via Deep Q-Learning (DQN). The goal of DQN is to predict the action-value function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ using a neural network. The input to the network are the states s with the output being the predicted action-value function q_π.

Under this setting, the state s can take on the form of continuous variables and may be of higher dimensions; and the actions taken may be of the form of a non-trivial function. These combined aforementioned factors make Artificial Neural Networks a suitable candidate for tackling these types of problems.

3.7 Feedforward Neural Networks

In this section we briefly introduce the key definitions and notions behind Feedforward Neural Networks (FNN). Neural networks are multidimensional composition functions that comprise of several layers; the input layer $I \in \mathbb{N}$, a set of hidden layers and an output layer $O \in \mathbb{N}$. More formally we define the structure for a FNN as follows [7, Definition 2.1]:

Definition 6. Let $I, O, r \in \mathbb{N}$. A function $\mathbf{f} : \mathbb{R}^I \to \mathbb{R}^O$ is an FNN with $r - 1 \in \{0, 1, ..\}$ hidden layers, where there are $d_i \in \mathbb{N}$ units in the i-th hidden later for any $i = 1, ..., r-1$ and activation functions of the form $\boldsymbol{\sigma}_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$ where $d_r = O$ if $\mathbf{f}$ has the form:

$$\mathbf{f} = \boldsymbol{\sigma_r} \circ \mathbf{L}_r \circ ... \circ \boldsymbol{\sigma_1} \circ \mathbf{L}_1, \tag{3.7.1}$$

where $\mathbf{L}_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ for any $i = 1, ..., r$ is an affine function of the form:

$$\mathbf{L}_i(\mathbf{x}) = W^i \mathbf{x} + \mathbf{b}^i, \mathbf{x} \in \mathbb{R}^{d_{i-1}} \tag{3.7.2}$$

parameterised by weight matrix $W^i = [W^i_{j,k}]_{j=1,..,d_i, k=1,...,d_{i-1}} \in \mathbb{R}^{d_i \times d_{i-1}}$ along with a bias vector $\mathbf{b}^i = (b^i_1, ..., b^i_{d_i}) \in \mathbb{R}^{d_i}$, with $d_0 = I$.

We provide a graphical representation of a Neural Network taken from [9, Figure 9.14 Chapter 9] to provide some context into FNN's:

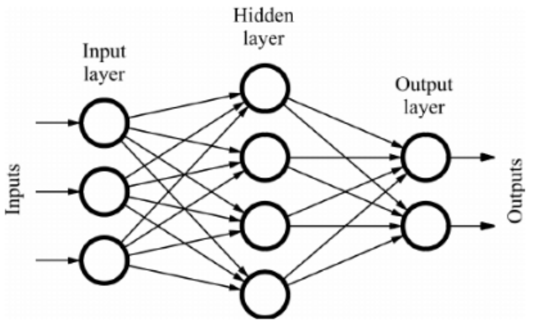


Figure 6: FNN with 3d input layer, 1 hidden layers and a 2d output layer

In our application and implementation of FNN's, we will be using the same number of units in each layer and the activation function we will be using is based on the ReLU function defined as follows:

$$ReLU(x) = \max\{x, 0\} \tag{3.7.3}$$

The training of a Neural Network involves fine tuning of the weights and biases, sampling disjoint mini batches across multiple epochs (the number of times a learning algorithm sees the complete data set). Neural Networks tend to work by minimising a loss function. Optimisers are the algorithms used in back-propagation and the goal of an optimiser is to find the optimum set of weights and biases which minimize the loss function. A commonly used method of minimisation is called stochastic gradient descent (SGD).

We briefly discuss the topic of Experience Replay as it is a fundamental concept that we will be using in our application of Deep Reinforcement Learning. We start of by defining a queue data structure with a fixed memory size M, which we call the replay buffer. The replay buffer adheres to a FIFO data structure in which episodes and experiences of the agent are fed into this container.

After a predetermined number of timesteps have elapsed, a subset (mini-batch) is sampled from the replay buffer and is then fed back into the network in order to train it. The experience of the agent at time t, are episodes that are of the form SARSA i.e. the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. The final step is to determine target vector Y_t which depends on the type of problem we wish to solve. Once this has been determined, we fit the network to the new observation using SGD. Further details on the SGD algorithm can be found in [7, Algorithm 3.1, Pg. 36].

3.8 Dueling DQN

This section on DDQN closely follows the work performed by [3]. We recall that we have state-value and action value functions following policy π as follows:

$$V_\pi(s) = \mathbb{E}[G_t | S_t = s] \tag{3.8.1}$$

$$Q_\pi(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a] \tag{3.8.2}$$

We now define the advantage value function as follows:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{3.8.3}$$

Intuitively one may interpret this advantage value function as depicting how advantageous it is selecting an action relative to others in a given state; since the Q value represents the value of selecting a particular action in a given state and the V value is the value at a given state regardless of the action taken.

The Dueling-DQN algorithm proposes a slight change in the final layer, and is split into two parts to estimate the advantage function A and state-value function V in order to predict Q. Since the actions that we take don't necessarily affect the state in which we are currently in - it is sufficient to learn only the state-value function. This very much fits the paradigm of our application of DQN since the target vega that we aim to sell, is independent of where the market may end up and we will see this later on in the report.

In the paper [3, Wang et al.] learning the Q-value by re-arranging the above relation i.e. Q=V+A, is not feasible as there are infinitely many solutions to this problem. The paper suggests a trick, which is to force the largest Q-value to be equal to the value V, thus making the highest value in the advantage function be zero and all other values negative. Thus we have the exact value for V, and we can calculate all the advantages from there, solving the problem. To add further clarity the training looks as follows:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \max_{a' \in |\mathcal{A}|} A(s,a;\theta,\alpha)) \tag{3.8.4}$$

The paper also suggests an alternative to using the maximum and replacing this quantity with the mean, which is the implementation we went with in the report:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)) \tag{3.8.5}$$

Further details can be found in the paper; and the stabilisation has been optimised without losing any rank information from an $\epsilon-$greedy policy.

4 Delta-hedging options with DDQN

In this theoretical setup we aim to train an agent to learn how to delta-hedge an ATM call option by learning the BS-Delta in a theoretical setting. This experiment acts as a stress-test and provided us with a foundation when applying Deep Reinforcement Learning in the trading setting. We formulate the problem as follows:

We start off by assuming that a stock price evolves with a GBM:

$$dS_t = rS_t dt + \sigma S_t dW_t$$

Where W_t is a standard Brownian motion. The solution to this SDE is:

$$S_T = S_0 \exp\left[(r - \frac{1}{2}\sigma^2)T + \sigma W_T\right]$$

We begin by discretising the time step so that incrementally we have:

$$S_{t_i} = S_{t_{i-1}} \exp\left[(r - \frac{1}{2}\sigma^2)(t_i - t_{i-1}) + \sigma W_{t_i - t_{i-1}}\right]$$

Now we aim to price the following 3M ATM call option with the following parameters:

Option parameters	**Values**
Initial Stock Price (S_0)	50
Strike	50
Maturity	90/365
Volatility	10%
Risk-free rate	0%

State Space:

In this framework we model the state space to contain a two-pair tuple; such that at time t, each state is comprised of the stock price at time t and the time to maturity left on the option denoted by $\tau = T - t$ so that we have:

$$s_t = \{S_t, \tau\} \in \mathcal{S}$$

Action Space:

As we wish to learn how to delta-hedge the option, we recall that a call option delta Δ_c satisfies the following inequality $0 \leq \Delta_c \leq 1$. We partition this interval into 101 uniformly sliced intervals so that an action set takes the form:

$$a_t \in \{0, 0.01, ..., 0.99, 1\}$$

Reward Function:

We deduce the reward function directly from the delta-hedging PnL. Thus taking the incremental change in the option price yields us with the following conditional expectation as a reward:

$$r_t = -|\mathbb{E}_t \left[a_{t_i+1}(S_{t_i} - S_{t_{i-1}}) - (C_{t_i} - C_{t_{i-1}})\right]|$$

Here the agent is penalised for an incorrect delta-hedge and rewarded more for a delta-hedge that is closer to the true delta of the option.

4.1 Dueling-DQN Setup & Algorithm

During the initial stages, the agent acts greedily and the maximum of all the Q-values will be chosen in accordance to the DDQN algorithm we specified in the previous section:

$$a* = \underset{a' \in \mathcal{A}(S_t)}{\arg\max}\, Q(s, a; \theta, \alpha, \beta)$$

The tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ is appended to the replay memory. We run a total of 10000 simulations for the stock price and respective option delta-hedge. After every 100 simulations have been run, the agent then makes a prediction from a random batch sample of 62 time steps to train the network. The target vector is then determined as follows:

$$Y_t(A_t) := R_t + \left(\gamma \underset{a \in \mathcal{A}(S_{t+1})}{\arg\max}\; Q(S_{t+1}, a)\right)$$

Where the Q-values are determined under the usual Dueling-DQN paradigm as defined earlier:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s, a'; \theta, \alpha)) \tag{4.1.1}$$

The network then minimises the Mean-Squared Error loss function between the target vector and predicted vector as follows:

$$MSE(Y, \hat{Y}) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.1.2}$$

The neural network is comprised of a 2-dim input layer, 2 linear hidden layers and a 1-dim output layer; each using the ReLU function as an activation function. The output layer equals the number of actions which can be taken in this case a singular number for the chosen delta-hedge target.

The codebase for building the reinforcement learning algorithm was heavily inspired by [8, PyTorch DQN Tutorial] and [11, Deep Q-Network Tutorial with PyTorch].

Algorithm 2: Dueling-DQN for Target Delta

```
1  Initialise DQN Agent, with algorithm parameters: $\alpha = 0.0005$ (learning-rate), $\gamma = 0.99$, initial-$\epsilon = 1.0$, $\epsilon$-decrement = -0.001, minimum-$\epsilon = 0.01$ score = -9999,
     Initialise Network architecture;
2  training-score = 0;
3  while $Episode \leq No.of Episodes$ do
4      Generate stock price path $S_t$; Initialize state $S_t$;
5      done = False;
6      t = 0;
7      while True do
8          $\epsilon$-greedy agent chooses delta-hedge action $A_t$;
9          Compute next stock price increment $S_{t+1}$ and reward $R_{t+1}$;
10         Add ($S_t$, $A_t$, $R_{t+1}$, $S_{t+1}$) to replay memory;
11         Increment memory-counter;
12         if memory-counter $\geq batch - sizememory$ then
13             Sample batch-size from replay memory;
14             Determine target vector Y for batch-size;
15             Fit Network of the DQN Agent;
16         if t == 360 then
17             Save PnL;
18             break;
19         if # Episode (mod) 100 = 0 then
20             Simulate 50 stock price trajectories;
21             Predict 3M ATM Delta;
22             Update aggregate score and store best model;
23         Update $\epsilon_{t+1} = \min(\epsilon - \epsilon$-decrement, minimum-$\epsilon$);
24         episode += 1;
```

In the below plot we have the training rewards per episode after every 100 simulations, and observe the aggregated training reward based on the sample of 50 stock prices. As we can see the agent effectively learns very well as the rewards begin to converge quickly towards 0 after 25 episodes where it maintains a stable training score profile.

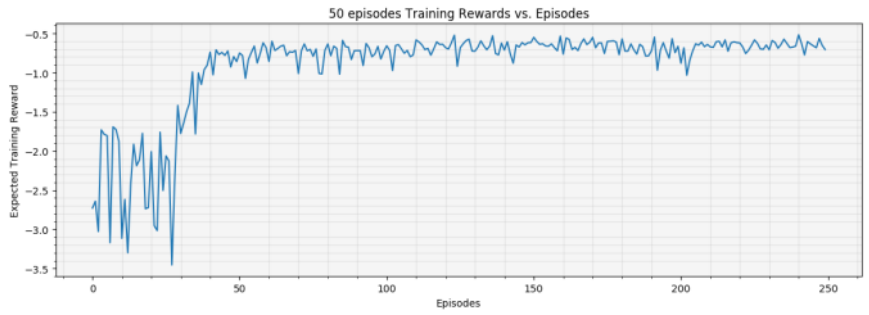


Figure 7: Episodic Training and Rewards gained by RL agent

Finally we load the best stored model and output the predictions for the delta of the 3M ATM Option. In addition to this, we wanted to test how good the model was at predicting OTM deltas. We sampled a range of stock prices and plot the corresponding delta in "moneyness". Overall the results were quite pleasing and there wasn't a huge scope of error in predicting OTM deltas despite being trained solely on an ATM option.

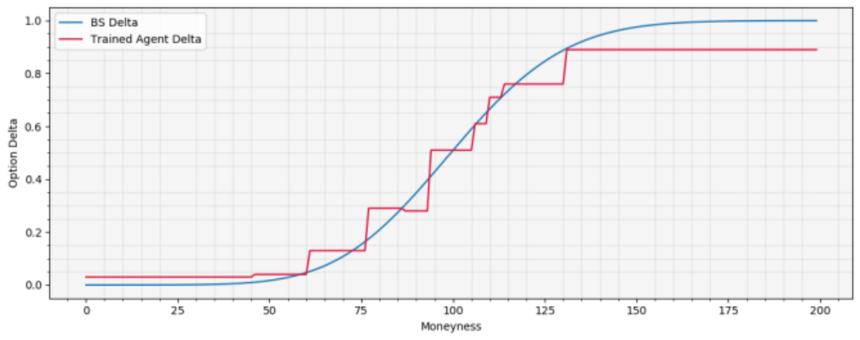


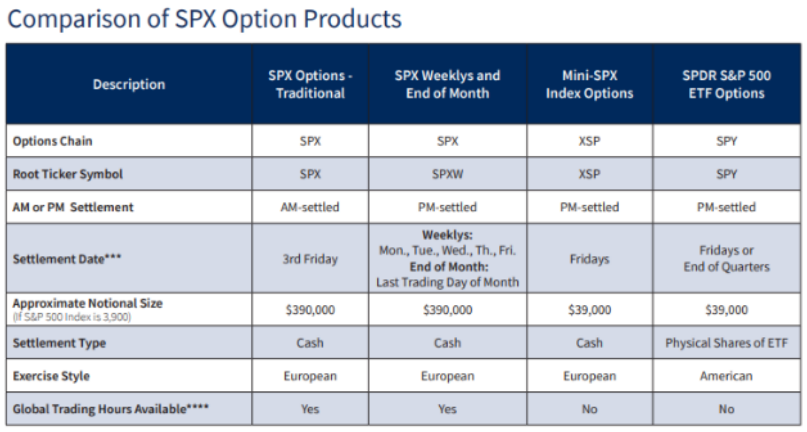
Figure 8: Final Output from RL agent on BS-Delta

This brief section gave us a practical introduction to the Dueling-DQN architecture and acted as a sanity check and the framework outlined provided us with a solid foundation for tackling Deep Reinforcement Learning problems. In the next section, we will introduce the concept of backtesting option strategies and we will be applying this network architecture to a much more complex trading setting.

5 Backtesting SPX Weeklys

5.1 Contract Specs and Backtester Setup

We begin by introducing the backtest setup as follows. we are trading SPX weeklies with the attached contract specs from CBOE:



Comparison of SPX Option Products

Description	SPX Options - Traditional	SPX Weeklys and End of Month	Mini-SPX Index Options	SPDR S&P 500 ETF Options
Options Chain	SPX	SPX	XSP	SPY
Root Ticker Symbol	SPX	SPXW	XSP	SPY
AM or PM Settlement	AM-settled	PM-settled	PM-settled	PM-settled
Settlement Date***	3rd Friday	**Weeklys:** Mon., Tue., Wed., Th., Fri. **End of Month:** Last Trading Day of Month	Fridays	Fridays or End of Quarters
Approximate Notional Size (If S&P 500 Index is 3,900)	\$390,000	\$390,000	\$39,000	\$39,000
Settlement Type	Cash	Cash	Cash	Physical Shares of ETF
Exercise Style	European	European	European	American
Global Trading Hours Available****	Yes	Yes	No	No

Figure 9: Overview of contracts specified by CBOE [5]

Our backtester aims to sell SPX Weeklys and End of Month options, as short gamma explanation is richest here.

Backtest Methodology

In this small section we provide a top level overview of the backtester and the components that are required to be specified in the trading configuration in running the backtester. The following attributes of the backtester can be found below:

1. We specify the strategy to be traded which involves the following attributes to be set in the trading configuration file:
   - The delta strikes and direction of the option strategy to be traded
   - The frequency of entering a new leg of the strategy (e.g. entering legs weekly)
   - The hold period of the strategy until unwind (e.g. hold to maturity or unwind after 3 days)
   - The frequency of delta-hedging (e.g. daily, weekly, at inception)
2. Compute the Strategy Index by amalgamating all of the individual leg PnL per day and summing this across all the overlapping legs (if any)

5.2 Constructing the Strategy Index and Performance Metrics

Strategy Index Calculation

In this section, we outline the calculation of the Strategy Index produced from the back-tester. The strategy index is comprised of the total PnL of the strategy which makes it a useful and effective tool in comparing the performance of several strategies. By default, we have a strategy index level which starts at 100 at t_0.

The index at time t (I_t) is computed as follows:

$$I_t = I_{t-1} + \Delta I_t, \tag{5.2.1}$$

where

$$\Delta I_t = \sum_{\text{Component i}} nU_t^i \left(C_t^i - C_{t-1}^i\right), \tag{5.2.2}$$

nU_t^i stands for the number of units of component i (either a future or option), and C_t^i is the \$ value of the that component, both observed at time t.

Performance Metrics

Upon construction of the strategy index, we then define a set of industry standard metrics to evaluate the performances of the strategy.

Definition 7. Sharpe/Information Ratio (SR/IR) - The Sharpe Ratio is defined as the annualised return of a strategy per unit risk (annualised volatility of the strategy), and is an industry standard unit of measure for the performance of a strategy. Mathematically it may be expressed as:

$$SR = \frac{\text{Ann. Return of ptf. } \Pi_t - \text{Avg. Risk-free Rate r}}{\text{Ann. Volatility of ptf. } \sigma_\Pi} \tag{5.2.3}$$

Definition 8. Maximum Drawdown (MDD) - The maximum drawdown of the strategy is defined as the maximum observed loss of the strategy from a running peak to trough. More formally it is computed as:

$$MDD = \min_t(\Pi_t - \max_{t'<t} \Pi_{t'}) \tag{5.2.4}$$

Definition 9. Calmar Ratio - Is another risk-adjusted return metric only this time we divide the annualised strategy returns by the max drawdown.

Definition 10. 5% cVaR (Expected Shortfall) - The 5% cVaR measures the tail-risk for a particular strategy by averaging the worst x% returns at the tails below a specified quantile level α. Mathematically we recall the definition from [10] as follows:

$$cVaR_\alpha(L) = \frac{1}{1-\alpha} \int_\alpha^1 q_u(L)du \tag{5.2.5}$$

5.3 Backtesting Volatility Carry Strategies

5.3.1 Systematic Short Strangle Backtest

We demonstrate the backtester by selling OTM vanilla option strangles (1 call and 1 put at different strikes where $K_c > K_p$) and we illustrate the payoff diagram for this short volatility strategy below taking S&P options as an example:

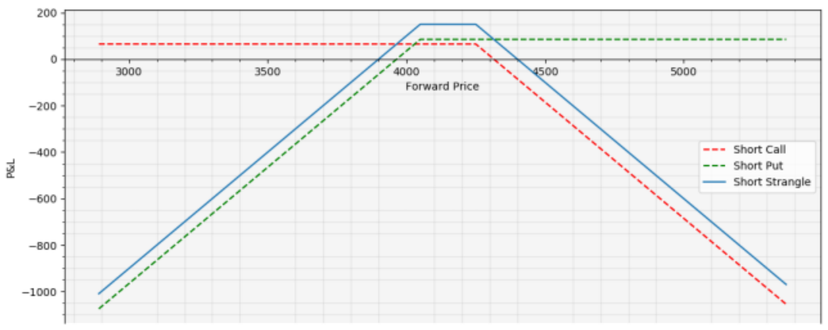


Figure 10: Short 1M 4050/4250 ESU2 Strangle (expiry: 16th Sep 2022) with futures ref: 4130.25 as of 1st August 2022

The PnL profile of this option strategy is that we collect the full premium if the forward price remains in-between the strangle strikes, however we have theoretically unlimited losses should the forward trade far beyond the strike range.

The backtest configuration for the systematic strategy of selling strangles is outlined as follows:

Backtest Parameters

- *Option Strategy:* We enter a short position in 40d strangles on SPX
- *Leg Entry Frequency:* We enter new strangle legs weekly
- *Leg Hold Period:* Each leg is held to maturity
- *Leg Sizing/Leverage:* 10bps target vega (to control leverage)
- *Delta-hedging Parameters:* Each leg is delta-hedged daily

Backtest Results

The results of the backtest are illustrated below:

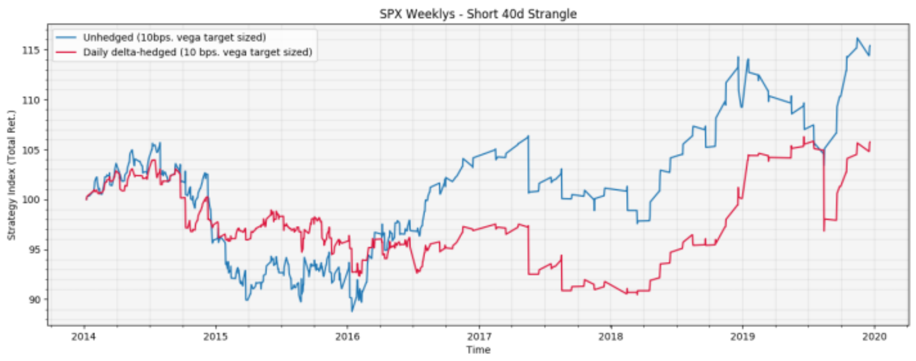


Figure 11: Systematic selling of 40d SPX strangles

Strategy	**Ann. Return %**	**Ann. Volatility %**	**Sharpe/Info Ratio**	**5% cVaR**
Unhedged	6.12	13.2	0.51	-0.07
Daily delta-hedged	3.24	10.1	0.24	-0.06

Table 2: Short 40d Strangle Backtest Metrics

We observe that in this backtest, that the unhedged strategy outperforms the daily-delta hedged counterpart on a risk-reward adjusted basis (Sharpe). This is because short gamma strategies tend to lock in losses at every delta-hedge (e.g. selling underlying futures low and buying high). However we comment on the fact that the variance of the PnL volatility is greatly reduced by daily-delta hedging (i.e. lower PnL volatility) which is in line with the theory of delta-hedging and trading options. The strategy has a positive annualised return over the 6 year of backtest history, indicating that on average we have positive carry in the short gamma strategy.

5.3.2 Systematic Short Synthetic Variance Backtest

In this section, we begin by illustrating selling synthetic variance via option strips. We outline the backtest parameters:

Backtest Parameters

- *Option Strategy:* We enter a short position in synthetic variance by selling strips of calls/puts from 50d to 5d
- *Leg Entry Frequency:* We enter new option strips weekly
- *Leg Hold Period:* Each leg is held to maturity
- *Leg Sizing/Leverage:* 10bps target vega (to control leverage)
- *Delta-hedging Parameters:* Each leg is delta-hedged daily

The results of the backtest are illustrated below:

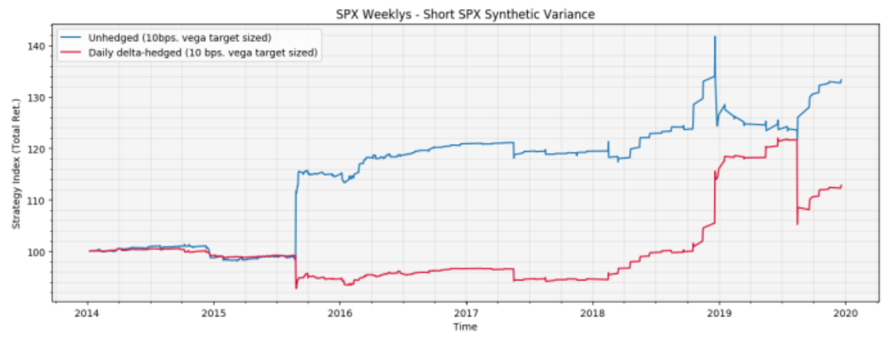


Figure 12: Systematic selling of SPX synthetic variance

Strategy	**Ann. Return %**	**Ann. Volatility %**	**Sharpe/Info Ratio**	**5% cVaR**
Unhedged	13.4	13.3	0.98	-0.04
Daily delta-hedged	5.23	13.2	0.42	-0.07

Table 3: Short Synthetic Variance Backtest Metrics

We briefly mention that again this strategy has an average positive return over the 6 year backtest history and has a positive carry profile. This is again in-line with our findings on the existence and harvesting the VRP that exists withing SPX options.

5.3.3 Stress-testing the Backtester

In order to stress test the backtester, we conducted the following simple experiment. On the day we entered a new option leg, instead of using the actual forward to price the options at time t, we instead ran a simulated forward price starting with the forward price at time t_0 using a discretised GBM. We then priced the options using this "simulated" forward price with a fixed constant implied volatility.

$$S_{t_i} = S_{t_{i-1}} \exp\left[(r - \frac{1}{2}\sigma^2)(t_i - t_{i-1}) + \sigma W_{t_i - t_{i-1}}\right]$$

This experiment served as a good stress-test when harvesting VRP. If the realised volatility from the simulated forward price was lower than the implied volatility of the options then we would have a positive VRP to harvest which meant that we should be harvesting vol risk premia and have a positive carry profile for the strategy; and the converse also holds. We would also expect that when the IV = RV we should have a relatively flat PnL profile over time. The following exhibits depict the result from this simulation:

<u>Flat vol: $\sigma_{RV} = \sigma_{IV} = 20\%$</u>

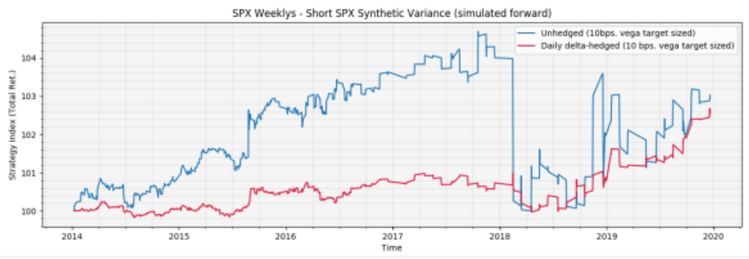


Figure 13: Flat VRP synthetic Var Backtest

Positive VRP: $\sigma_{RV} = 10\%, \sigma_{IV} = 20\%$

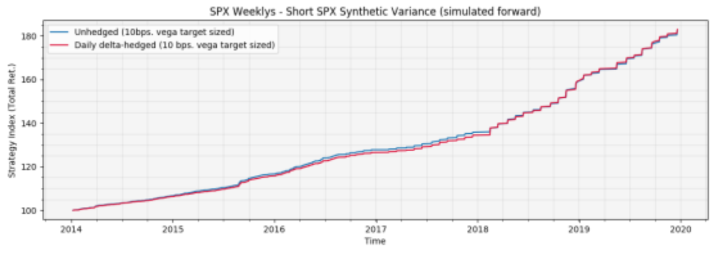


Figure 14: Positive VRP synthetic Var Backtest

Negative VRP: $\sigma_{RV} = 30\%, \sigma_{IV} = 20\%$

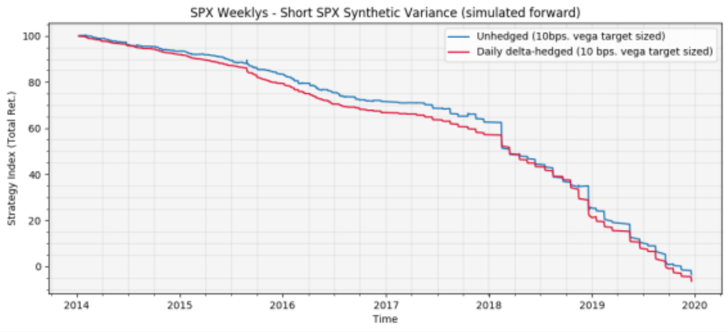


Figure 15: Negative VRP synthetic Var Backtest

To summarise this section, the results of the backtest were inline with our expectations meaning that we saw a positive PnL carry for short gamma strategies when VRP was positive and the converse held for when VRP was negative. This gave us the confidence that our backtester was robust and running as expected.

5.4 Trained RL Agent - Short Synthetic Variance Backtest

In this section we trained an RL agent to learn the optimal portfolio vega to short when performing the same short synthetic variance backtest we saw in (5.3.2). Using a similar architecture to the training of the agent to delta-hedge we now proceed to define the RL setup.

RL Setup

Under this setup we recall that the strategy index is a function of the individual leg units involved in the strategy (options and futures in our case); taking data with a time horizon up to time t. We adopt the RL terminology and call the index strategy a *policy*. We define our policy π to be:

$$\pi : (f(nU_t^i, C_t^i, ...)) \rightarrow \Delta nU_t^i$$

Here the function $f()$ allows us to pass deterministic transformations of units and prices as features to the model, and also provide additional market data and portfolio metrics such as BS-Delta/BS-Vega which we will see later on.

In our formulation of the problem, we restrict the strategy to using pricing information available from time $t-1$; thus:

$$\pi : (f(nU_t^i, C_{t-1}^i, ...)) \rightarrow \Delta nU_t^i$$

We then allow for the policy to produce an output which we can then transform into incremental units. Thus it makes sense as an approach to have the model output incremental "bet sizes" which can be seen as leverage controllers in the strategy - to deleverage and leverage up where appropriate. If we denote function $g()$ as this transformation then our policy π becomes:

$$\pi : (f(nU_t^i, C_{t-1}^i, ...)) \rightarrow g^{-1}(\Delta nU_t^i)$$

Here we see that $f(nU_t^i, C_{t-1}^i, ...)$ corresponds to a state s_t and $g^{-1}(\Delta nU_i^t)$ corresponds to an *action* a_t, so that we have $\pi :\rightarrow a_t$ and $g(a_t) = \Delta nU_t^i$. We will now define the spaces and action sets in more detail.

<u>State Space:</u>

We define our states to be the total portfolio vega at different bumped levels of futures prices denoted as $\hat{F}_t$:

$$\hat{F}_t = \{(F_t^{-20\%}, F_t^{-15\%}, F_t^{-10\%}, F_t^{-5\%}, F_t^{0\%}, F_t^{+5\%}, F_t^{+10\%}, F_t^{+15\%}, F_t^{+20\%})\}$$

So that the state s_t takes the form:

$$s_t = \{(\nu_t^{-20\%}, ..., \nu_t^{+20\%})\} \in \mathcal{S}$$

Where $\nu_t^{X\%}$ represents the total portfolio BS-Vega (i.e. the sum of all the individual leg BS-Vega contained in the strategy at time t) evaluated at each of the different bumped futures prices. More formally we define this 9-tuple as the Vega Ladder of the portfolio at time t, held by the strategy.

Action Space:

For the DQN agent, we choose a predetermined bound of min/max total portfolio vega that the agent is allowed to short. By keeping the total portfolio vega range fixed, the agent is prevented from selling unlimited quantities of vega when VRP is rich; thus the net vega exposure of the strategy is capped, meaning that the strategy isn't subject to any direct systemic risk.

We define the portfolio vega bounds below:

Min Short Portfolio Vega	**Max Short Portfolio Vega**
0.15%	2.25%

And the action set is split into 101 uniformly sliced elements ranging from the minimum/maximum bound of target vega specified above.

Reward Function:

The most obvious choice for defining the reward function is linking it directly to trading PnL. The DQN Agent is more penalised for negative PnL and rewarded for trades with a positive PnL. We define the reward as follows:

$$r_t = -\left|\mathbb{E}_t\left[a_{t+1}\left(\sum_{i=0}^{n} nU_t^i\left(C_{t+1}^i - C_t^i\right)\right)\right]\right|$$

Where a_{t+1} denotes the action i.e. amount of vega to sell and nU^i, C^i denote the number of units and price of component respectively. When the agent acts greedily, the maximum of all the Q-values will be chosen in accordance to the DDQN algorithm we specified earlier:

$$a* = \underset{a' \in \mathcal{A}(S_t)}{\arg\max}\, Q(s, a; \theta, \alpha, \beta)$$

The tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ is appended to the replay memory and once one week of data has elapsed (i.e. 5 trading days) a random batch sample of 32 trading days is chosen from the replay memory to train the network. The target vector is then determined as follows:

$$Y_t(A_t) := R_t + \left(\gamma \underset{a \in \mathcal{A}(S_{t+1})}{\arg\max}\, Q(S_{t+1}, a)\right)$$

Where the Q-values are determined under the usual Dueling-DQN paradigm as defined earlier:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s, a'; \theta, \alpha)) \tag{5.4.1}$$

And the network then minimises the Mean-Squared Error loss function between the target vector and predicted vector as follows:

$$MSE(Y, \hat{Y}) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{5.4.2}$$

The neural network is comprised of a 9-dim input layer, 2 linear hidden layers and a 1-dim output layer; each using the ReLU function as an activation function. The output layer equals the number of actions which can be taken in this case a singular number for the chosen vega-target to be sold.

We describe the architecture for the applied RL problem with the following RL diagram:

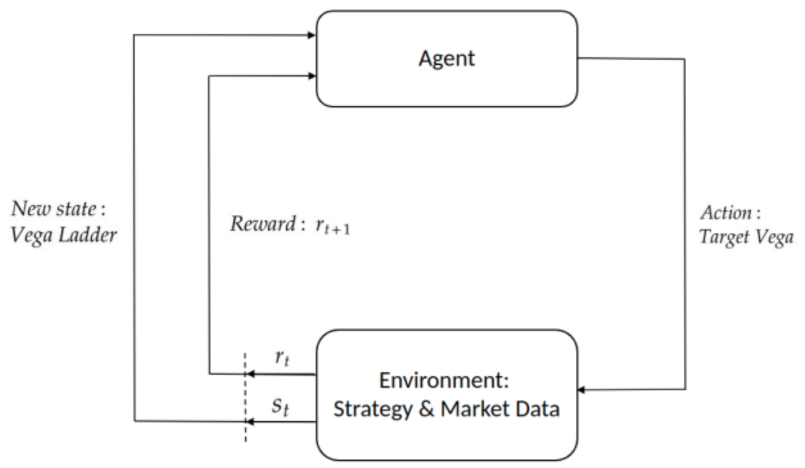


Figure 16: RL Diagram for Target Vega

DDQN RL Backtesting Algorithm

Due to the lack of data points based on the history of the SPX weekly options; we were restricted to only training the agent on data every week between 1st September 2009 to 1st January 2014. Ideally we would have liked to have trained the agent on at least 10 years worth of data history as well as extending the training to 50 agents sampling the best 5 performing agents to jointly make the trading decision (called the ensemble method).

The implementation of the algorithm was crucial and a fine balance between managing a flexible data pipeline (i.e. manipulation and storage of data) as well as managing computational efficiency of the code with acceptable run time.

Algorithm 3: Dueling-DQN for Target Vega

```
1  Initialise DQN Agent, with algorithm parameters: $\alpha = 0.0005$ (learning-rate), $\gamma =$
     $0.99$, initial-$\epsilon = 1.0$, $\epsilon$-decrement = -0.001, minimum-$\epsilon = 0.01$ score = -9999,
     Initialise Network architecture;
2  training-score = 0;
3  foreach episode do
4      Initialize Vega Ladder State $S_t$;
5      while step of episode do
6          $\epsilon$-greedy agent chooses action $A_t$ (target vega position) to sell;
7          The next Vega Ladder state is determined from next trading day $S_{t+1}$;
8          Compute Reward $R_{t+1}$;
9          Add ($S_t$, $A_t$, $R_{t+1}$, $S_{t+1}$) to replay memory;
10         Increment memory-counter;
11         if memory-counter (mod 5) $\equiv 0$ then
12             Sample batch-size from replay memory;
13             Determine target vector Y for batch-size;
14             Fit Network of the DQN Agent;
15         if training-score $\geq$ score then
16             Store model and update Network parameters and score;
17         Update $\epsilon_{t+1} = \min(\epsilon - \epsilon$-decrement, minimum-$\epsilon$);
18         episode += 1;
19
```

As we can see below, the training of the agent did seem to converge to a somewhat sub optimal solution and learned well particularly during 70-80 episodes in which the learning rate seemed to converge and stabilise. As mentioned previously, the lack of data points in SPX weekly options meant that the agent may not be able to learn as efficiently how to cope with certain tail events that arising from systemic risk in the market place.

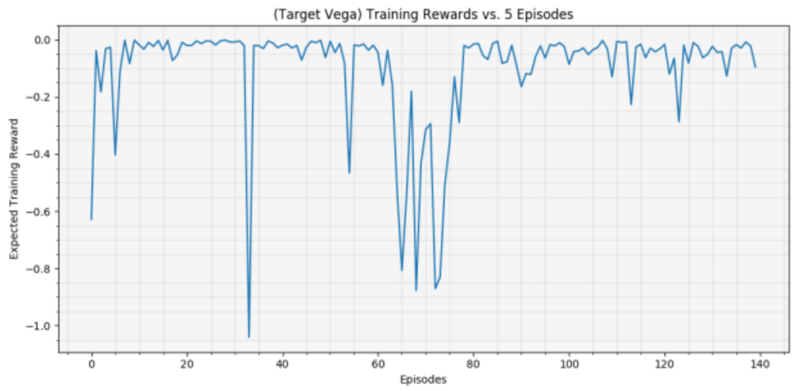


Figure 17: Training rewards for RL agent on a weekly basis over 140 episodes

In the below diagram we outline the empirical distribution of the target Vega sold, and we can see from the learning process that the agent was a lot more conservative in the target portfolio Vega sold given that the distribution was slightly more skewed to the right. This means that the overall strategy has less leverage and will have a much lower volatility than the constant Vega counterpart; which is favourable amongst investors in yielding a much more stable PnL profile.

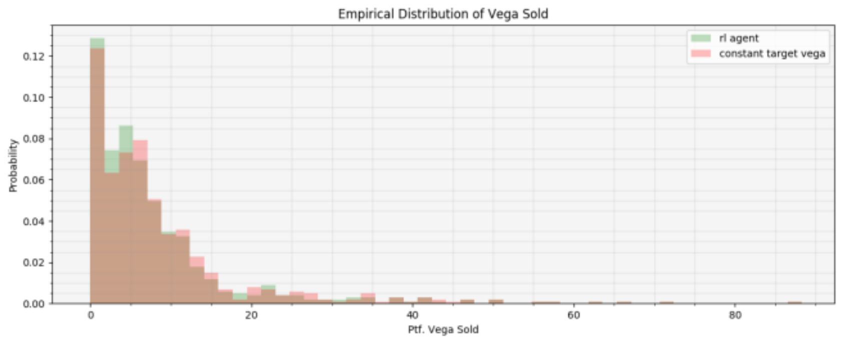


Figure 18: Distribution of target Vega sold

The final results of the backtest are illustrated below:

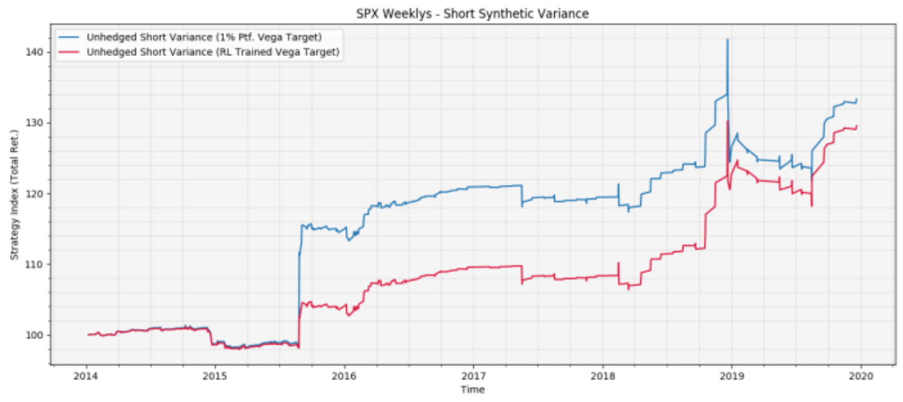


Figure 19: Strategy Index of RL trained Synthetic Variance vs. Constant Vega Target Backtest

Strategy	**Return**	**Vol**	**Sharpe**	**5% cVaR**	**Max DD.**	**5% Calmar**
1% constant ptf. Vega	0.13	0.13	0.98	-0.04	-0.14	0.90
RL Trained target Vega	0.11	0.09	1.24	-0.03	-0.09	1.23

Table 4: Strategy Performance Metrics

We can see from the PnL profile that the strategy had a far more stable PnL profile over time; with a $\tilde{30}$% reduction in overall PnL volatility; and a significant increase in Sharpe Ratio ($\tilde{2}$7% improvement). The max draw down was also reduced with a lower leverage; however this came as a consequence as the overall annualised strategy returns were slightly lower than the constant Vega counterpart.

6 Conclusion

In this project we were able to get a glimpse into applying Deep Reinforcement Learning techniques to trade non-linear risk premia utilising a combination of Neural Networks and Reinforcement Learning algorithms. Deep Reinforcement Learning has had a significant presence in financial markets in recent years due to it's successes and some of it's applications are currently being pioneered at Bulge Bracket Investment Banks such as Bank of America, J.P. Morgan and Societe Generale.

This project began by covering many topical aspects of volatility, including it's various definitions, some misconceptions on volatility as well as introducing various methods to trade volatility to harvest alternative risk premia. Systematic trading strategies benefit greatly from a time perspective since market timing of when to enter a trade is less of a concern and execution slippage is widely reduced. One drawback to the backtester is that a trading cost model was not implemented i.e. including execution costs/rebates from trading listed Equity options.

It is also worth mentioning that the exchange data set that was used in the project wasn't entirely the best and granular. It required a lot of cleaning/parsing and a lack of data points for listed SPX Weeklys meant that the RL agent training could only be performed on a small subset of granular data points, when a ten year history for training would have been preferred. Additionally having intraday futures data would have greatly helped since it meant that we could have delta-hedged intraday which would ultimately lead to a larger scope for training of the RL agent.

Despite the computational challenges of modelling and trading a neural network in this setting, we were able to demonstrate a successful VRP harvesting systematic trading strategy of selling synthetic variance on a weekly basis with an agent controlling the overall leverage (target vega) of the portfolio on each day.

There is significant scope for future work on this project; different neural network architectures (e.g. DDPG), experimenting with different layer sizes and activation functions in the neural network, training multiple agents and aggregating the best-performing agents via the ensemble method to name a few.

Overall this project was incredibly challenging, particularly with the complexity of building an option back tester from scratch as well as embedding and implementing a reinforcement learning agent to work in conjunction with the back tester; turned out to be a an extremely strenuous task which involved deep knowledge of Python and design patterns to optimise code production and keep execution run times within capacity of the machinery used.

References

[1] Sebastien Bossu et al. *Introduction to Variance Swaps.* Willmott Magazine, 2006.

[2] Sebastien Bossu et al. *Just what you need to know about Variance Swaps.* J.P. Morgan, 2005.

[3] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning.* https://arxiv.org/abs/1511.06581, 2015.

[4] G. Bakshi and N. Kapadia. *Delta-Hedged Gains and the Negative Market Volatility Risk Premium.* http://dx.doi.org/10.1093/rfs/hhg002, 2003.

[5] *CBOE SPX Index Options Factsheet.* CBOE, 2022.

[6] Vladimir Lucic. *Volatility Notes.* https://papers.ssrn.com/sol3/papers.cfm?abstract$_id =$ 3211920, 2019.

[7] Dr. Mikko Pakkanen. *Math97231 - Deep Learning Lecture Notes.* Dept. Of Mathematics, Imperial College London, 2020.

[8] *Pytorch DQN Tutorial.* https://pytorch.org/tutorials/intermediate/reinforcement$_qlearning.html$, 2022.

[9] Andrew Barto Richard S. Sutton. *Reinforcement Learning An Introduction 2nd. Edition.* 1992. ISBN: 9780262193986.

[10] Daniel Schwartz. *MATH97108 - Quantitative Risk Management.* Dept. Of Mathematics, Imperial College London, 2020.

[11] Chao De-Yu. *Deep Q-Network, with PyTorch.* https://towardsdatascience.com/deep-q-network-with-pytorch-146bfa939dfe.

Source Code

All of the project source code has been uploaded to github in a private repository with the following URL: `https://github.com/nileshram/RL_VRP`